



President's Message

by Tom Kinahan, N1CPE

Just as the Flea Market season reopens for the year, keep in mind that you can list gear for sale, or gear that you are interested in buying here in the **Spark Gap**. Send your info to the editor, Jon Freeman, KA1NDX at his home address listed elsewhere. Also, please feel free to put together some words on your favorite aspect of Ham Radio and get them to Jon to get it in the **Spark Gap**.

- We need a volunteer to organize our Field Day efforts this year, and be the clearinghouse for plans.
- Don't forget the June VHF contest coming up.
- Get your nominations for next years club offices in.

Nominations for Offices

We're currently looking for nominations for club offices in the upcoming club year. The present officers are listed in this issue. Please consider nominating someone you want or yourself for any office. Send nominations to Wellesley A.R.S. Election Committee, 211 Washington St. Wellesley, Ma. 02181, or to the chairman of the committee, C.J. McDonald, 48 Hale St. Newton, Ma. 02164.

Novice Class

The Novice class is progressing quite nicely. To date five individuals have taken their exams and passed. One of the new novices, Peter Borner, put in a great effort operating WITKZ (third party) in the Novice Roundup. Good Show!

Jim, KC1BB taught principles and Ohm's Law. Two monday night classes were snowed-out. (no great surprise) Jim and Frank, NG1I gave up two Saturday mornings for the make-up classes.

I taught circuits and antennas. The last class I taught was one of the Advanced classes. There sure is some big difference between teaching an Advanced class and a Novice class. Not only is the material easier to teach, but the students are a bit more trusting.

Frank has done his very best to not only teach the morse code in a thorough manner, but instill an enjoyment of CW as a communications mode.

MEETINGS THIS MONTH:

March 5, 1987 at 7:30 P.M.
March 19, 1987 at 7:30 P.M.
211 Washington St., Wellesley

The club appreciates everyone who has worked hard to help the Novice class along. Please be encouraging and friendly to the new folks.

If you feel strongly about the growth and survival of the hobby, please stop by one Monday night and give the students some words of advice.

Vern Valero, ND1Z

Repeater News

A hearty **Thank You** to all those who have contributed to the repeater fund. We're doing well, but still can use more contributions for the repeater antenna. These are the people that have contributed thus far :

Dave McLellan, N1DFP	Bill Furbish, N1EHM
Lou Marotta, N1EHI	Vern Valero, ND1Z
Liz Valero, KA1PNA	Stan Waugh, N1EHG
George Joubert, KA1JIP	
Craig Brock Hank Ostrowski, VE3CGL	
George Goldin, KA1LQN	Ed Gallagher, WB1FCZ
Mort Cohan, KA1IU	Art Thompson, KA1ENP
Al Graf, W1QQP	Dick Weissel, WA1HRV
George Vedic, KA1LOO	C.J. McDonald, WB1BUM
Ged Wear, NF1Z	Tom Kinahan, N1CPE
Roger Smith, W1EOA	Bill Baker, KA1NKJ
Rita Baker, N1DDI	

FCC Exam !!

The Wellesley A.R.S. will conduct FCC Exams for all classes of amateur licenses on May 2nd at 10 am at the Wellesley Red Cross. For additional information Contact Dan Brevik, KZ1D at 460-9059.

NEWS FLASH !

Novices will get voice and digital privileges much as the ARRL had proposed starting on March 21.

Specifically, on 10 meters, Novices will get digital and SSB privileges at 200 watts maximum output power. They will also get 220 MHz privileges between 222 and 224 MHz in all

modes at 25 watts including the use of repeaters. At 1200 MHz novices will get all mode privileges for a portion of this band with 5 watts maximum power output. More specifics to come.

220 MHz Threat

The FCC has proposed the reallocation of the frequencies 220-220 MHz from hams to some commercial interests. This doesn't affect the new Novice privileges but takes some spectrum from the rest of us. Stay tuned to the amateur press and to W1AW bulletins for more info.

Wanted - Dead or Alive :

Preferably dead - Heathkit HW16 transceiver for use by W.A.R.S. novices. Contact Roger Smith, W1EOA.

New Ticket !

Congratulations to Liz Valero, XYL of Vern ND1Z, for getting her Novice call recently, KA1PNA - KA1 Pretty Normal Amateur. She reports her first QSO (with NF1Z) went flawlessly. When is Sara going to get on with her license? Stay tuned...

Calendar ! Calendar !

- *March 5* : W.A.R.S. Meeting. Civil Air Patrol presentation by Cal Maddox, WA1ZGH
- *March 19* : W.A.R.S. Meeting. Meteor Scatter voice systems presentation by John Herman.
- *May 2* : FCC Exam, 10 A.M. at Wellesley Red Cross.

The Spark Gap

Official Journal of the Wellesley Amateur Radio Society

Editor:

Jon Freeman
17 East Meadow Road
Lowell, Ma. 01854



The Spark Gap is a monthly publication of the Wellesley Amateur Radio Society, its sole purpose is to keep its members abreast of current happenings, FCC rule & regulation changes and to list equipment offered for sale or trade by members. Members are encouraged to contribute to the Spark Gap and may send their articles to the editor's address or to the club address. Items printed in the Spark Gap may be reprinted by other Amateur Radio publications provided the author and the Spark Gap are given credit.

The Wellesley Amateur Radio Society is a non-profit organization dedicated to the promotion of Amateur Radio through instruction, fellowship, demonstration and participation. Annual membership in the society is \$10 and the club year runs from September 1 through August 31. Amateur Novice through Advanced classes are conducted regularly at no cost. To enroll in a class or gain information about the Society in general, write :

The Wellesley Amateur Radio Society

211 Washington Street

Wellesley Hills, Ma. 02181



New Members

We wish to welcome two new members this month, both members of the Novice class. **Craig Brock** from Lexington, and **Peter Borner**, of Hopedale. Craig is hoping to take the Novice exam soon we all wish him luck. Peter has already taken his Novice exam, passed, is waiting for his call, and has even made his first QSO's during Novice Roundup as a third party at W1TKZ!

March Presentations

Cal Maddox, WA1ZGH will speak on Civil Air Patrol at the March 5, 1987 W.A.R.S. meeting. John Herman of GTE Strategic Systems Division in Westborough, will speak on applications of Meteor Scatter voice systems during the March 19th W.A.R.S. meeting.

For Sale —

- Heathkit HW-101 80 - 10 Meter SSB and CW rig with power supply AND spare 6146 final tubes \$ 250 —
- Commodore VIC 20 with 16K memory expansion, VIC FORTH program language and many extras \$ 125 —

Contact Jim Charboneau KC1BB

Day Phone: 273-4844

OFFICERS

President

Tom Kinahan, N1CPE, 898-2749

Vice President

Rita Baker, N1DDI, 481-0863

Recording Secretary

Roger Smith, W1EOA, 235-6239

Corresponding Secretary

Arthur Thompson, KA1ENP, 235-5509

Treasurer

C.J. McDonald, WB1BUM, 332-6091

Fund Raising

David Kent, N2AWG, 875-2126

Operating Activities Chairman

Bob Drapeau, KT1K, 883-1132

Education Chairman

Repeater Committee Chairman

Paul Davis, N1EGC, 646-3599

Making Computers Work

by Vern ND1Z

Now that many of us have bought home computers, how do we get them to work for us? The answer is that there are two options. One may buy programs that have already been written and tested, or write them independently. The purchase option is the quickest way to get a program to run, but costs are high (\$20 to \$ 100 per program). Typically one is unable to change the purchased program. This installment of the series will investigate what is involved in writing a program on one's own.

A distinction is made in the computer world between the electronics and circuitry and the actual programs. The electronics is known as Hardware and the collection of programs is known as Software.

Writing software is really a process of many actions. One must think things through and proceed carefully to produce a program with the desired results. In fact, the process can be decomposed into several steps that are the following: Problem Definition, Feasibility Study, Solution Definition, Design, Coding, Debugging, Testing, and Maintenance.

The word 'Problem' is used in the engineering sense of the word. A computer is a tool for solving problems. A flight simulator program solves the problem that exists when an inexperienced pilot is not ready to fly, but needs to learn how to react to aeronautical conditions. When writing a program, one must clearly understand the problem to be solved. Perhaps by thinking about it, one might realize that the problem really isn't a problem at all.

It makes sense to consider the feasibility of

writing a program to perform a desired job. Some applications do not lend themselves to a computer solution or would not work well on a particular computer. For example, a home computer that does not have a disk drive (but does have a tape cassette) would not be a good candidate for a contest logging program because the operator would need to start and stop the cassette recorder too often for the program to save the logging data. This would leave less time for making contest QSO'S.

Once the problem is understood and a confident decision is made to use the computer to solve it, the solution must be cooked-up. This is by far the most difficult step, but surely the most important. Many beginning programmers overlook defining the solution and jump directly to the writing process (coding) only to be disappointed later. Defining a solution requires understanding something about the application. If one is going to write a general ledger program, it would be a good idea to understand something about accounting. Eventually, a very high level approach comes to mind. If we want to write a program to compute averages, then the general solution is to add the numbers and divide by the number of values. We remember this from high school math.

The design step takes the solution statement and amplifies it to greater detail. There are a number of techniques that have been developed over the years to make this step easier. There are such things as flow charts, data dictionaries, data-flow diagrams, and many others. The right tool to use depends upon the nature and size of the application. The larger the size of the program and number of people writing it, the more the need for these design tools. This author prefers a notation consisting of a series of numbered

statements in outline form of how the program will run. Here is how the 'Averages' program would appear:

```
1.0 Initialize the running sum and
    counter to 0
2.0 DO read a value from the
    keyboard UNTIL the character
    'X' is given
    2.1 ADD number to running sum
    2.1 ADD 1 to the counter
ENDO (end-of-loop)

3.0 IF the counter is nonzero
    3.1 THEN divide the running
        sum by the count. Print
        this as the result average.
    3.2 ELSE Print the message:
        'NO AVERAGE BECAUSE NO VALUES
        WERE INPUT'
```

The flow of control in a program should follow a few simple rules. It should run from top to bottom. There should be one entrance into a program and one exit from a program. All processing should conform to the following constructs, or building blocks:

1. Procedure Construct: Just a series a simple tasks to perform once. Most computer languages have a facility known as subroutines that allow the programmer to execute a specific procedure in different places without having to keep repeating the statements.
2. DO Construct: Performing a series of tasks many times in a loop. The loop may happen a specific number of times, or its continuance or ending may be contingent upon a specific condition.
3. IF THEN ELSE Construct: IF a certain condition holds, THEN do a certain action; ELSE do some other specific action.
4. CASE Construct: Given a certain input, branch to one of possibly many

different process constructs and then return to the bottom of the CASE area. For example, a program may do 8 or 9 different activities depending upon a user input value. The procedure for each activity is contained within the Case block.

Also note that our design above contains two 'housekeeping' activities. One is to initialize the running sum and count to zero. This ensures that we start clean. The other is to check for errors on the part of the user. What would happen in 3.0 if the user only input 'X' to signal end of inputting without actually entering any values? We would have divided by zero, which is illegal. The program would have failed with an error.

The error messages must be clearly understood by the user. Everyone makes mistakes and our program must be tolerant of this fact. Each error message must tell the user exactly what is wrong and what steps must be taken to correct the problem. There is nothing more disconcerting than mistyping an input only to have the program print "FATAL ERROR!!!". Programs that are 'User-Friendly' encourage rather than discourage.

The coding phase involves implementing the design into a specific computer language. A computer language is a set of statements and rules for programming a computer. There are many languages, some of which are best suited for specific applications. Each statement of a language has a specific purpose and may be translated into one or more machine instructions. The compiler or interpreter program is a built-in service that translates from the high-level statements to low-level machine instructions. For example, the following language statement that adds one to a summation might be translated into the following machine code on an Intel 80186 processor:

```

LET SUM = SUM + 1 (The value in SUM has
                   one added to it in the
                   high level language,
                   BASIC)

0000 1010      (These are the series of
0000 0001      binary instructions that
0000 0000      might be generated by a
0000 0000      compiler or interpreter.
0000 0101      Remember that a computer
0000 0000      can only execute binary
0000 1000      instructions. This is
1010 0011      the machine code.      )
0000 0000
0000 0000

```

Clearly it is much easier to program in a higher level language than in machine language!

The most popular language for home computers is the Beginners All-Purpose Symbolic Instruction Code, otherwise known as BASIC. Developed in the mid-1960's at Dartmouth College, it was designed to be quick and easy to learn. Although BASIC does not have the flexibility of some of the other languages, we will use it to demonstrate the averaging program example. Below is an implementation of the design we did above for computing averages:

```

10 REM
20 REM AVERAGES PROGRAM FEB 5, 1987
30 REM COMPUTES THE AVERAGE OF A VARIABLE
40 REM NUMBER OF VALUES
50 REM
60 REM TO USE: KEEP INPUTTING ALL THE NUMBERS
70 REM AND THEN INPUT X
80 REM
90 REM INITIALIZE RUNNING SUM (S) AND COUNTER (C)
100 LET S = 0
110 LET C = 0
120 PRINT 'AVERAGES PROGRAM .....'
130 PRINT
140 PRINT 'INPUT ALL NUMBERS TO BE AVERAGED'
150 PRINT ' WHEN DONE INPUT X'
160 PRINT
170 REM DO READ VALUES UNTIL 'X'
180 REM
190 INPUT A$
200 IF A$ = 'X' THEN 270
210 LET S = S + VAL(A$)
220 LET C = C + 1
230 GO TO 190
240 REM
250 REM IF THE COUNTER IS NOT 0 THEN COMPUTE

```

```

251 REM AVERAGES
260 REM ELSE GIVE ERROR MESSAGE
270 IF C <> 0 THEN 330
280 PRINT 'NO AVERAGES BECAUSE NO VALUES
      WERE INPUT'
300 GO TO 100
310 REM AVERAGE IS SUM DIVIDED BY COUNT
320 REM
330 D = S/C
340 PRINT 'AVERAGE VALUE IS... ',D
350 END

```

Space does not permit a detailed explanation of the BASIC language, but a few comments about the program are in order. The statements REM mean 'Remark' or 'Comment'. They are not executable statements but are reminders to the programmer of what is going on in the program. We input the value from the keyboard and if it is an 'X', then finish inputting. If it isn't 'X', then we add the value to the running sum and increment the count. When the user finally types 'X', we determine if any numbers were input. If not, an error message is output and the user is given another chance. Finally, after some numbers have been input, the sum is divided by the count and the result printed.

This program contains a blatant error. It was a problem that was overlooked in the design phase and will be propagated through all the remaining phases. It is left as an exercise for the reader to determine what the problem is and to propose a solution. Good Luck!

The debug phase consists of running through all the considered input cases and correcting problems as they are observed. Some computers have debugger programs that help with this task. Debuggers have the ability to stop program execution at various points and observe what is happening in the program.

The testing phase is really a self-check. All the known input cases are run in an organized manner. If a program is very large it is a good

idea to write a test plan that lists each expected input and output. Below is the output of the averaging program that is listed above:

AVERAGES PROGRAM

INPUT ALL NUMBERS TO BE AVERAGED
WHEN DONE INPUT X

!3
!6
!45
!8
!33
!9
!X
AVERAGE VALUE IS ... 17.3333333333

AVERAGES PROGRAM

INPUT ALL NUMBERS TO BE AVERAGED
WHEN DONE INPUT X

!X
NO AVERAGES BECAUSE NO VALUES WERE INPUT
AVERAGES PROGRAM

INPUT ALL NUMBERS TO BE AVERAGED
WHEN DONE INPUT X

!3
!4
!5
!X
AVERAGE VALUE IS ... 4

Now that the program is in everyday use, we enter the maintenance phase. Software is 'Soft' and capable of being changed. One will wish to correct errors and perhaps change things slightly to suit better usage. This small change activity is known as maintenance.

Lest the reader conclude that he or she knows all there is to know about programming, be forewarned! All computer systems are different to some degree. A program that runs on an Apple II might not work properly on a Commodore 64 because BASIC itself may work differently on both systems. Furthermore, the details of how to create, test, and run a program varies greatly

from system to system. There is no substitute for reading the system booklets and spending a great deal of time experimenting and becoming familiar with the computer system.

This is an overview of what is involved in programming, or making computers work for us. The next part of this series will overview some of the amateur radio software programs that can be purchased.

Computer Glossary

This small glossary has been compiled for your reference in reading the series of Spark Gap articles on computers. It is by no means complete, but should help clarify some of the terminology.

- **Address** - Each value in main memory has an address just as each house on a street does. Data is stored and recalled from specific numbered addresses.
- **Algorithm** - A 'plan of attack' in solving a problem with the use of a computer. For example, an algorithm for finding an average value would be: 'Add all values and divide by the number of values'.
- **Application** - A use or specific purpose for using a computer. For example, Joe Ham has purchased a Commodore 64 to balance his checkbook and run RTTY. The applications are accounting and RTTY.
- **ASCII** - American Standard Computer Information Interchange. A character code, or standard representation for displaying data. In the ASCII code, the letter 'A' is represented in memory by the binary number 635.
- **Binary** - The representation of data in memory as ones and zeroes using

digital circuits. For example, in the binary system, the number 7 would be represented as 111 which really is $(1 \times 2^2) + (1 \times 2) + 1 = 7$.

- **Bit** - A single cell in memory that can take on either a 1 or 0 value. Each fixed number of bits (usually 8) usually has its own unique address.
- **Boot** - The process of putting a program into a computer that has just been powered up and getting that program started.
- **Byte** - A fixed number of bits with a unique address. Most computers have an 8 bit byte.
- **Character** - A fixed number of bits that have a value conforming to a standard code (such as ASCII). When the character code value is sent to an I/O device such as a monitor, a specific letter or number is printed on the screen. If the number 635 were output to the monitor, the letter 'A' would be printed.
- **Compiler** - A program that looks at the contents of a program file (a program that the user has written) and translates the high level statements into low level computer instructions for execution. (running the program) The compiler produces machine level instructions, but does not insert memory addresses. The Loader program has knowledge about the memory structure of the computer and inserts address values.
- **Constant** - A memory location that contains a certain value and that value never changes during program execution.
- **Data Structure** - An organization of data that is of interest. For example, the membership list might be organized

into a list of records, one for each member. Each record might contain name, call sign, address and telephone number.

- **Disk** - A storage device that holds programs and data that are not currently in use. The bits are stored magnetically and the computer accesses the desired data by moving the disk to the correct spot.
- **Execution** - The act of running a program. (nobody dies)
- **File** - A location on a disk for storing data. Each file has its own name and can be accessed by the user. The list of filenames is known as a directory. Each file contains data of specific interest. For example, the file WAS might contain a list of states that have been worked and the file DXCC might contain the list of countries worked.
- **Graphics** - A special way of operating monitors such that pictures appear on the screen.
- **Hardware** - The section of the computer that includes all circuitry and external I/O elements.
- **I/O** - (Input/Output) The means of sending data into the computer or receiving data from the computer and interpreting it.
- **Instruction** - (Machine Instruction) A single task that the computer can perform at its lowest level.
- **Interpreter** - A program that looks at each statement of a program file, translates that statement, and executes it before going on to the next statement. Interpreters are very popular for home computers because it is quick to run and debug programs. However,

programs executed by interpreters tend to run slowly.

- **Interrupt** - A break in the normal execution of a program usually caused by an I/O device. The computer quickly performs some action (such as accepting a character from the keyboard) and then continues the program.
- **Language** - The set of rules that determine how to write a program. The most popular language for home computers is BASIC.
- **Logic** - The digital circuits that form computer building blocks.
- **Loader** - A program that places memory addresses into the output file of the compiler. Generally, programs that have been processed by both the compiler and the loader are now ready to be executed.
- **Memory** - The section of hardware in which data and programs are stored.
- **Networking** - A means of tying together different computers in geographically distributed areas for exchanging data.
- **Object** - The file that contains the list of instructions produced by a compiler program.
- **Packet** - A networking scheme in which large streams of data are sent from one computer to another in bite-sized chunks.
- **Pointer** - A memory location that contains the address of some program section or data structure.
- **Procedure** (Subroutine) - A modular section of a program that can be used at any time of program execution

simply by requesting, or 'calling' that section. For example, a procedure to add 2 values may be called several times in a program that computes averages.

- **Protocol** - The set of rules that computers use in networking for exchanging data.
- **Queue** - A data structure in which the data that was first received is acted upon first.
- **Runfile** - (Image) The file that contains the list of instructions with the correct memory addresses produced by the loader program. A runfile can be placed into memory and run.
- **Simulation** - The use of a computer to mimic a real life occurrence. For example, pilot trainees use a computer simulator to learn how to react to different aeronautical conditions.
- **Software** - The section of the computer that includes programs and data.
- **Stack** - A data structure in which the data that was last received is first acted upon.
- **Structured Programming** - A programming technique in which flow of control follows certain rules to enable the program to be easily understood and modified.
- **Table** - A list of values in memory.
- **Tape** - An I/O device that stores programs and data on a reel of magnetic tape in a sequential manner.
- **Terminal** - An I/O device that enables data to be printed on a TV-type screen and entered from a keyboard.

The following members were left out of the previous membership list due to press deadlines :

Wellesley A.R.S. Members, 1/7/87, continued

PETER BORNER, 190 LAURELWOOD DRIVE, HOPEDALE, MA 01747	N	473-8606
CRAIG BLOCK, 15 LAKE ST. LEXINGTON, MA 02173		863-3517
BOB DOWNER, 99 PHILIP STREET, MEDFIELD, MA 02052	T N1COY	359-5023
GEORGE J. GOLDIN, 14 LLEWELLYN ROAD, WEST NEWTON, MA 02165	N KA1LQN	224-6110
GARY IRVING, 48 CHESLEY ROAD, NEWTON, MA 02159	T KA1HDR	969-9487
KENNETH W. MC NAIR, 105 BLAKE STREET, NEEDHAM, MA 02192	G N1DEZ	444-5317
HANK OSTROWSKI, P.O. BOX 1088, SIMPSONVILLE, S.C. 29681	A VE3CGL	803-233-0290
THOMAS RILEY, 50 HALE STREET, NEWTON, MA 02164	N WB1BUN	244-3239
RALPH H. VACCA, 45 THORNTON ROAD, WALTHAM, MA 02154	E W1KF	899-0490
GEORGES VEDIE, 85 BLAKE STREET, NEEDHAM, MA 02192	N KA1LQO	444-0405

The Spark Gap



**WELLESLEY
AMATEUR RADIO
SOCIETY**

211 Washington Street
Wellesley, Ma. 02181



FIRST CLASS MAIL